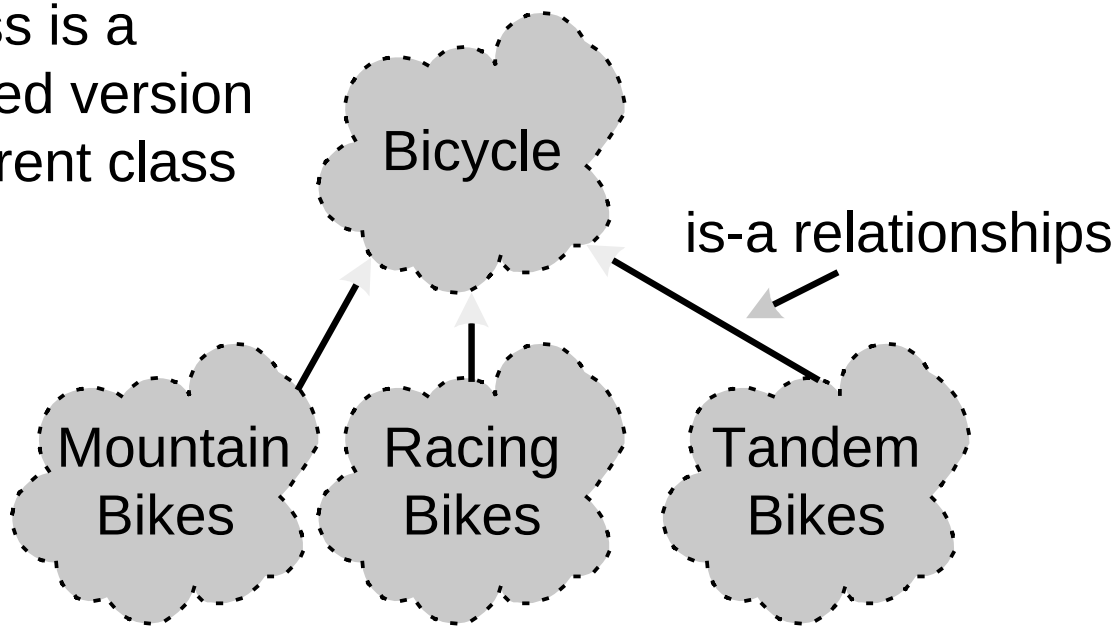
J. P. Cohoon and J. W. Davidson

Inheritance

Mechanism for deriving new classes uses existing classes as bases

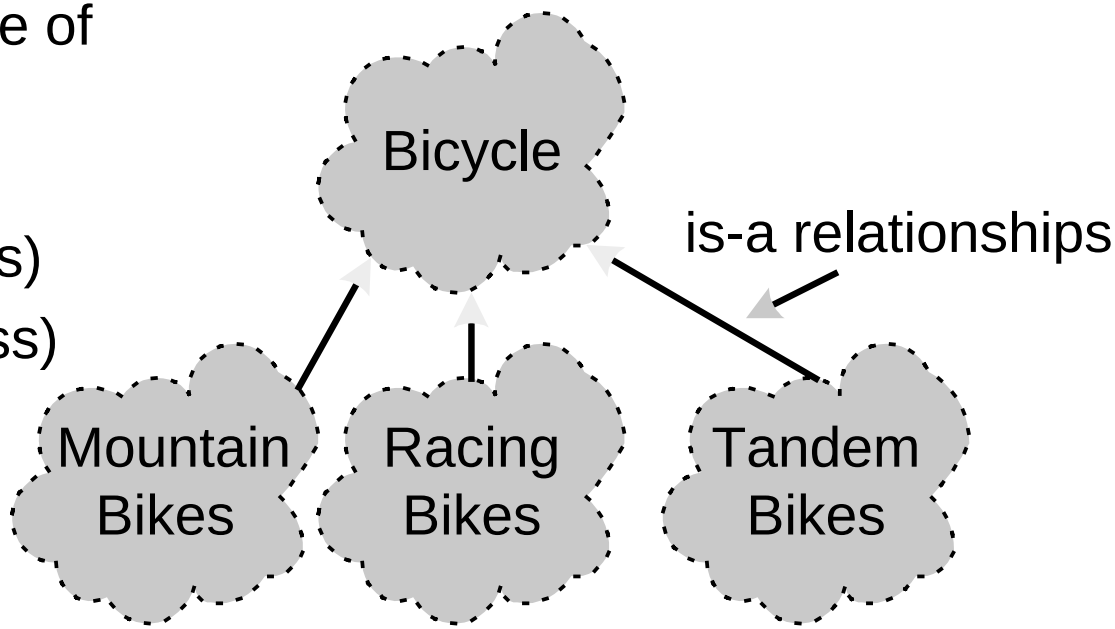
Inheritance

- Ability to define new classes of objects using existing classes as a basis
  - The new class inherits the attributes and behaviors of the parent classes
  - New class is a specialized version of the parent class



Inheritance

- A natural way to reuse code
  - Programming by extension rather than reinvention
  - Object-oriented paradigm is well-suited for this style of programming
- Terminology
  - Base class (superclass)
  - Derived class (subclass)



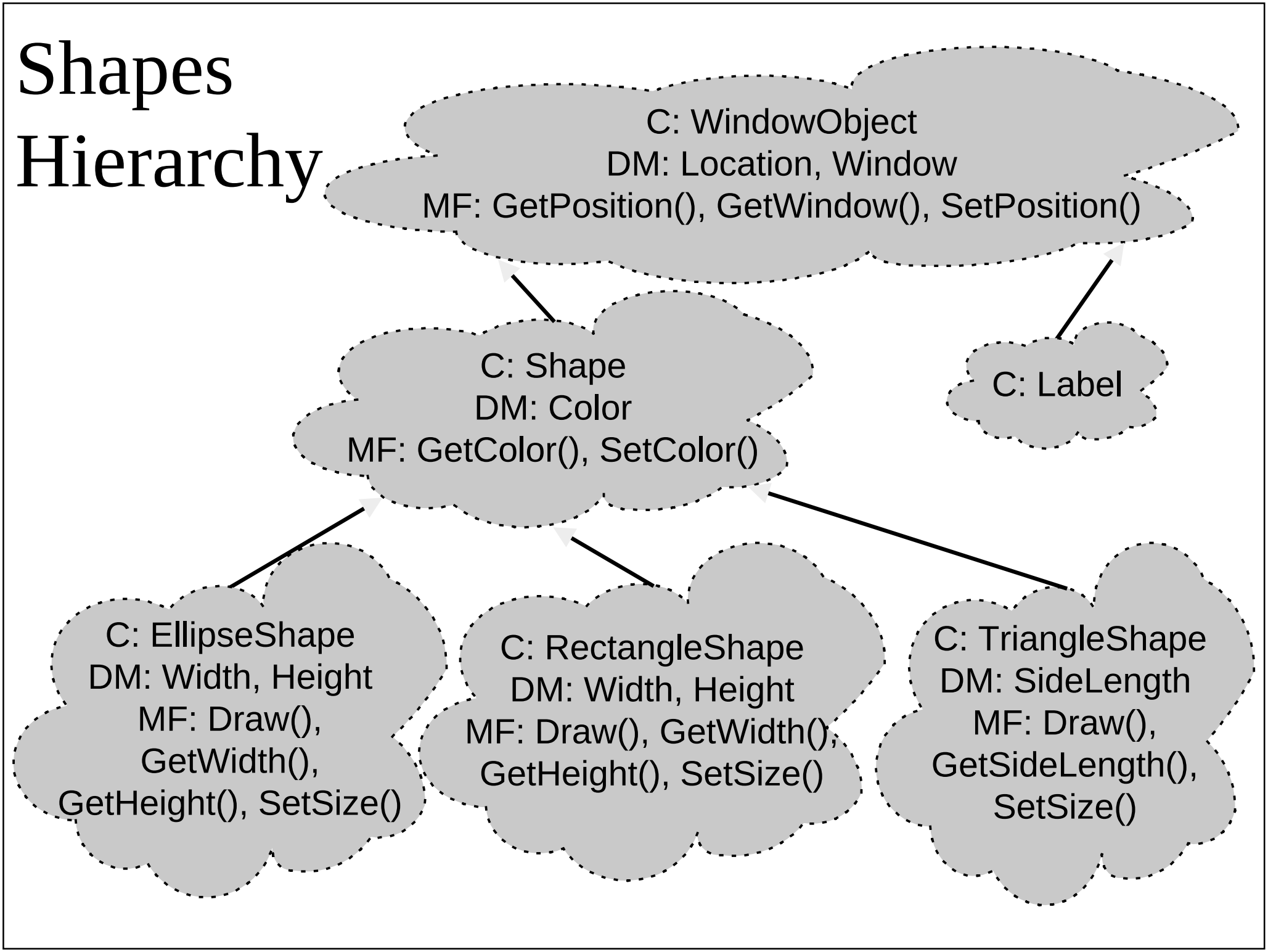
Before Inheritance

```
class RectangleShape {
  public:
      RectangleShape(SimpleWindow &W,
       float XCoord, float YCoord, const color &Color,
       float Width, float Height);
      void Draw();
      color GetColor() const;
      void GetSize(float &Width, float &Height) const;
      void GetPosition(float &x, float &y) const;
      float GetWidth() const;
      float GetHeight() const;
      SimpleWindow& GetWindow() const;
      void SetColor(const color &Color);
      void SetPosition(float x, float y);
      void SetSize(float Width, float Height);
  private:
      SimpleWindow &Window;
      float XCenter;
      float YCenter;
      color Color;
      float Width;
      float Height;
};
```

Before Inheritance

```
class CircleShape {
   public:
       CircleShape(SimpleWindow &W, float x, float y,
        const color &Color, float Diameter);
       void Draw();
       color GetColor() const;
       float GetSize() const;
       void GetPosition(float &x, float &y) const;
       SimpleWindow& GetWindow() const;
       void SetColor(const color &Color);
       void SetPosition(float x, float y);
       void SetSize(float Diameter);
   private:
       SimpleWindow &Window;
       float XCenter;
       float YCenter;
       color Color;
       float Diameter;
};
```

Shapes Hierarchy



Class WindowObject

```
class WindowObject {
  public:
      WindowObject(SimpleWindow &w,
       const Position &p);
      Position GetPosition() const;
      SimpleWindow& GetWindow() const;
      void SetPosition(const Position &p);
  private:
      SimpleWindow &Window;
      Position Location;
};
```

WindowObject Constructor

```
WindowObject::WindowObject(SimpleWindow &w,
 const Position &p) : Window(w), Location(p) {
 // No body needed
}
```

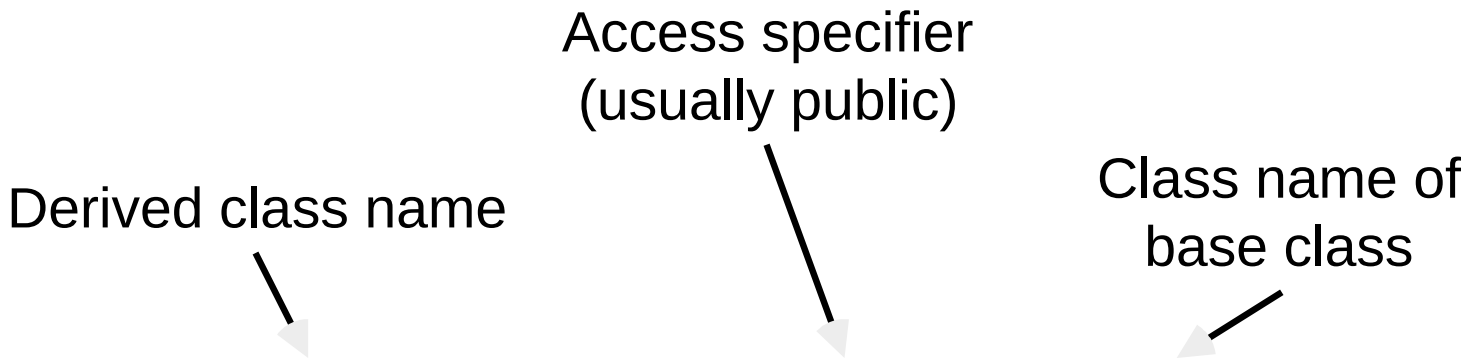
WindowObject Inspectors

```
Position WindowObject::GetPosition() const {
   return Location;
}

SimpleWindow& WindowObject::GetWindow() const {
   return Window;
}
```

WindowObject Mutator

```
void WindowObject::SetPosition(const Position &p){
   Location = p;
}
```

Defining a Derived Class




```
class DerivedClass : public BaseClass {
  public:
      // public section
   ...
  private:
      // private section
   ...
};
```

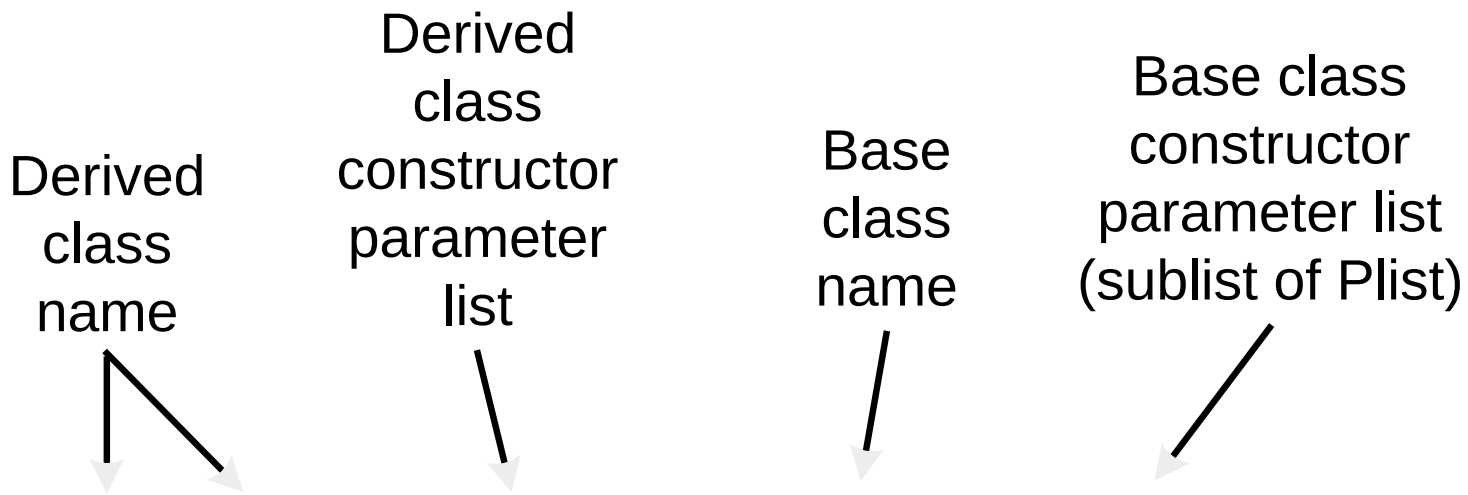
Declaring a Derived Class

Read this as "Shape is a kind of WindowObject"

```
class Shape : public WindowObject {
  public:
      Shape(SimpleWindow &w,
       const Position &p,
       const color &c = Red);
      color GetColor() const;
      void SetColor(const color &c);
  private:
      color Color;
};
```

Shape inherits WindowObject members Window, Location, GetPosition(), GetWindow(), and SetPosition()

Implementing a Derived Class



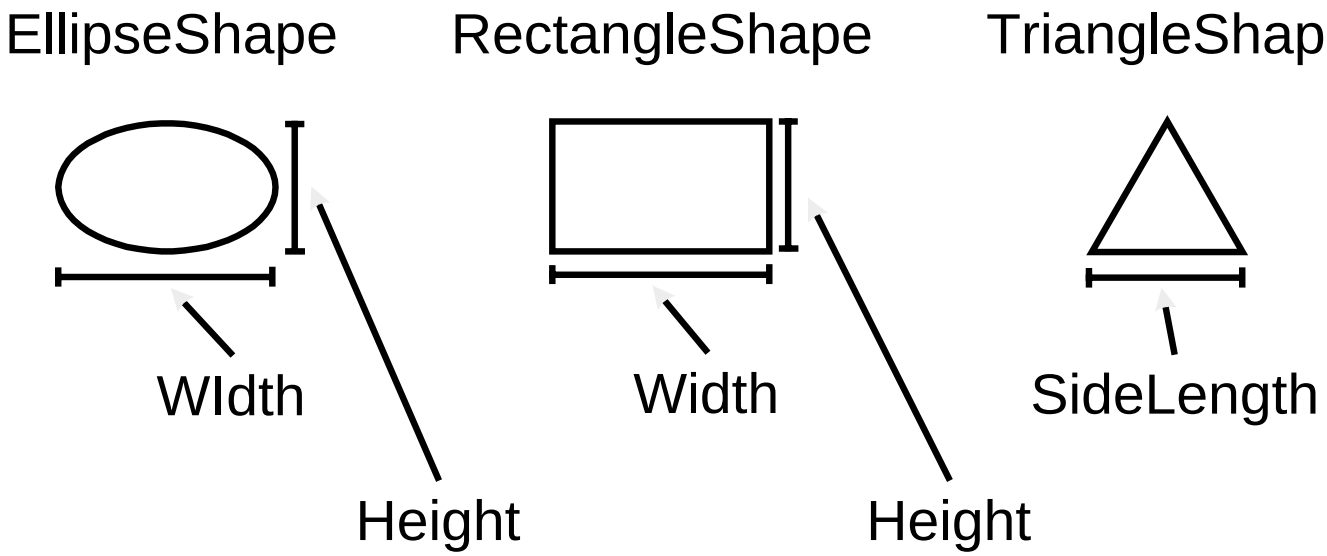

```
DClass::DClass(PList) : BClass(BList), DMbrList {
   // Body of derived class constructor
   ...
};
```

Class data member initialization list

Implementing a Derived Class

```
Shape::Shape(SimpleWindow &w, const Position &p,
 const color &c) : WindowObject(w, p), Color(c) {
 // No body needed
}

color Shape::GetColor() const {
   return Color;
}

void Shape::SetColor(const color &c) {
   assert(c >= 0 && c < MaxColors);
   Color = c;
}
```

Basic Shapes



TriangleShape

```
#include "shape.h"
class TriangleShape : public Shape {
  public:
      TriangleShape(SimpleWindow &w,
       const Position &p, const color &c = Red,
       float slen = 1);
      float GetSideLength() const;
      void SetSize(float slen);
      void Draw();
  private:
      float SideLength;
};
```

EllipseShape

```
#include "shape.h"
class EllipseShape : public Shape {
  public:
      EllipseShape(SimpleWindow &w,
       const Position &Center,
       const color &c = Red, float Width = 1,
       float Height = 2);
      float GetWidth() const;
      float GetHeight() const;
      void Draw();
      void SetSize(float Width, float Height);
  private:
      float Width;
      float Height;
};
```

RectangleShape

```
#include "shape.h"
class RectangleShape : public Shape {
   public:
       RectangleShape(SimpleWindow &w,
        const Position &Center, const color &c = Red,
        float Width = 1, float Width = 2);
       float GetWidth() const;
       float GetHeight() const;
       void Draw();
       void SetSize(float Width, float Height);
   private:
       float Width;
       float Height;
};
```

TriangleShape::Draw()

```
void TriangleShape::Draw() {
  const float Pi = 3.1415;
  const Position Center = GetPosition();
  const float SLength = GetSideLength();

  // Compute c, distance from center of triangle
  // to the top vertex, and a, the distance from
  // the center to the base of the triangle
  float c = SLength / (2.0 * cos(30 * Pi / 180.0));
  float a = tan(30 * Pi / 180.0) * .5 * SLength;
```

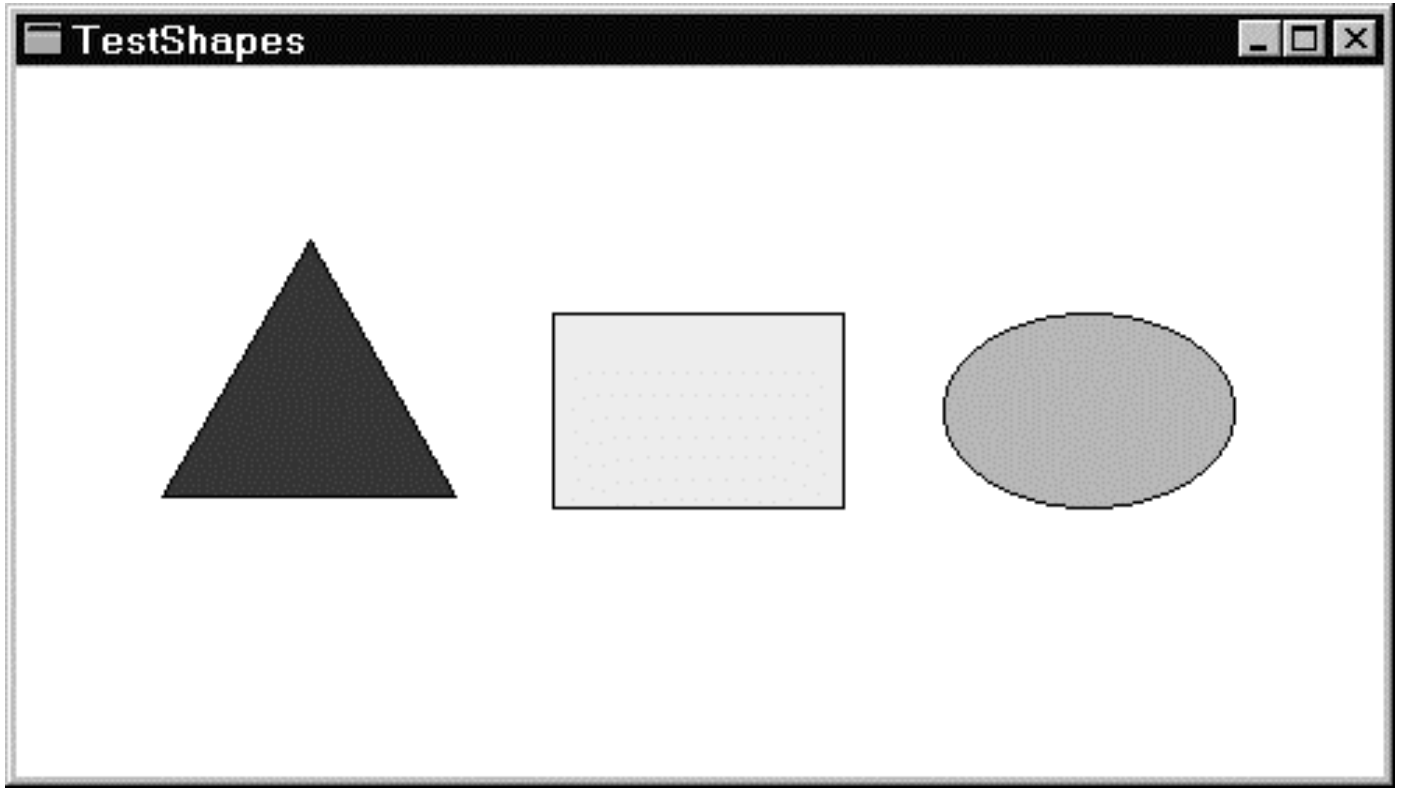
TriangleShape::Draw()

```
    // Create an array containing the positions of
    // the vertices of the triangle
    vector Position TrianglePoints[3];
    TrianglePoints[0] = Center + Position(0, -c),
    TrianglePoints[1] = Center
     + Position(-.5  * SLength, a);
    TrianglePoints[2] = Center
     + Position(.5 * SLength, a);
    // Draw the triangle
    GetWindow().RenderPolygon(TrianglePoints, 3,
     GetColor(), HasBorder());
}
```

Using Shapes

```
#include "rect.h"
#include "ellipse.h"
#include "triangle.h"

SimpleWindow TWindow("TestShapes", 17.0, 7.0,
   Position(4.0, 4.0));

int ApiMain() {
   TWindow.Open();
   TriangleShape T(TWindow, Position(3.5, 3.5),
    Red, 3.0);
   T.Draw();

   RectangleShape R(TWindow, Position(8.5, 3.5),
    Yellow, 3.0, 2.0);
   R.Draw();

   EllipseShape E(TWindow, Position(13.5, 3.5),
    Green, 3.0, 2.0);
   E.Draw();

   return 0;
}
```

Fun with Shapes



Cleaning Up

```
int ApiEnd()
   TWindow.Close();
   return 0;
}
```

Inheritance and Member Access

```
class SomeClass {
   public:
       void MemberFunction();
       int MyPublicData;
   protected:
       int MyProtectedData;
   private:
       int MyPrivateData;
};

void SomeClass::MemberFunction() {
   MyPublicData = 1;     // access allowed
   MyProtectedData = 2;  // access allowed
   MyPrivateData = 3;    // access allowed
}
```

Inheritance and Member Access

```
void NonMemberFunction() {
   SomeClass C;
   C.MyPublicData = 1;       // access allowed
   C.MyProtectedData = 2;    // illegal
   C.MyPrivateData = 3;      // illegal
}
```

Inheritance and Member Access

```
class BaseClass {
   public:    int MyPublicData;
   protected: int MyProtectedData;
   private:   int MyPrivateData;
};
class DerivedClass : public BaseClass {
   public: void DerivedClassFunction();
   // ...
};
void DerivedClass::DerivedClassFunction() {
   MyPublicData = 1;     // access allowed
   MyProtectedData = 2;  // access allowed
   MyPrivateData = 3;    // illegal
}
```

Controlling Inheritance

Inheritance Type	Base class member access	Derived class member access
public	public	public
	protected	protected
	private	inaccessible
protected	public	protected
	protected	protected
	private	inaccessible
private	public	private
	protected	private
	private	inaccessible